CHAPTER 6

UNIT V

Clustering Techniques

Syllabus

Hierarchical Clustering, Expectation maximization clustering, Agglomerative Clustering - Dendrograms, Agglomerative clustering in Scikit- learn, Connectivity Constraints.

Introduction to Recommendation Systems : Naïve User based systems, Content based Systems, Model free collaborative filtering-singular value decomposition, alternating least squares.

Fundamentals of Deep Networks : Defining Deep learning, common architectural principles of deep networks, the building blocks of deep networks.

Clustering is an unsupervised learning as classes are not known. It groups the object or data elements without having advance knowledge of group definitions. These groups or partitions of data after clustering are called clusters. Clusters have high inter similarity and low intra similarity.

6.1 Hierarchical Clustering

The set of data objects is decomposed hierarchically using certain criteria. In hierarchical clustering, algorithms either follow the top down or bottom up approach.

Hierarchical clustering technique (Basic algorithm)

1. Calculate the distance and find a distance matrix or proximity matrix
2. Consider each data point or object as a cluster.
3. Repeat.
4. Combine the two closest clusters.
5. Update the proximity matrix.
6. Until only a single cluster remains.

Note : Proximity matrix is symmetric, meaning that the numbers on the lower half of the diagonal will be the same as the numbers on the top half of the diagonal.

Agglomerative Clustering

In Bottom up approach, every object is considered to be a cluster and in subsequent iterations they are merged into a single cluster. Therefore, it is also called as **Hierarchical Agglomerative Clustering (HAC).**

Divisive Hierarchical Clustering

In this data objects are grouped in a top down manner. Initially, all objects are in one cluster. Then the cluster is subdivided into smaller and smaller pieces, until each object forms a cluster on its own or until it satisfies certain termination conditions as the desired number of clusters are obtained. Divisive methods are not generally available, and rarely have been applied.

Difference between Agglomerative and Divisive Hierarchical Clustering

Sr. No.	Agglomerative	Divisive
1.	Start by considering each individual element as a cluster.	Start with only one cluster by combining all elements.
2.	At every step, clusters are merged based on closest pair until a single or k clusters are left	At every step, the cluster is divided or split until each cluster contains a single element or k clusters are left.

Advantages

1. This algorithm is simple and gives an output as a hierarchy
2. The structure obtained is easy to understand and more informative.
3. There is no need to pre-specify the number of clusters

Disadvantages

1. Merging and splitting is critical once the clusters are formed, as undo is not possible. Every time it performs on newly generated clusters.
2. If decision to merge and split is wrong, then low quality clusters are formed
3. Scikit-learn implements only the agglomerative clustering as the complexity of divisive is higher than Agglomerative and has similar performance of Divisive approach.

6.2 Expectation Maximization Clustering

- The Expectation Maximization (EM) algorithm can be applied to bring forth the best theory for the distributional parameters of some multi-modal data.
- The best hypothesis for the distributional parameters is the maximum likelihood hypothesis – the one that maximizes the probability that this data we are looking at comes from K distributions, each with a mean m_k and variance $sigma_k^2$
- The Expectation Maximization (EM) Algorithm deal with missing labels by alternating between two steps :
  1. **Expectation (E) :** Fix model and estimate missing labels.
  2. **Maximization (M) :** Fix missing labels (or a distribution over the missing labels) and find the model that maximizes the expected log-likelihood of the data.

General EM Algorithm

The alternate steps until model parameters don't change much :

- **E step :** Estimate distribution over labels given a certain fixed model.
- **M step :** Choose new parameters for model to maximize expected log-likelihood of observed data and hidden variables.

Formal Setup for General EM Algorithm

Let $D = \{x^{(1)},....x^{(n)}\}$ be n observed data vectors.

Let $Z = \{z^{(1)},....z^{(n)}\}$ be n values of hidden variables (i.e. the cluster labels)

Log-likelihood of observed data given model :

$$L(\theta) = \log p(D|\theta) = \log \sum_{Z} p(D,Z|\theta)$$

Note : both θ and Z are unknown.

Where theta is a vector of unknown parameters.

In clustering, K (number of clusters) are not known initially. We can produce two results for the clustering :

- **Hard clustering :** In this method , data object or observation i belongs to only one cluster, so clusters can be produced like { C1, C2, ...Ck}.
- **Soft clustering :** The method says that data object or observation is more likely to belong to one of the K cluster by producing a probability distribution :

$$\mathbb{P}[X_i \in C_k] = \gamma_{k,i} \qquad \text{with } \sum_{k=1}^{K} \gamma_{k,i} = 1$$

From Fig. 6.2.1, we can observe that hard clustering is possible on the left side figure. But right side figure is not clear, so only based on probability samples can be classified to one of the class. This is the purpose of soft clustering. Algorithms like hierarchical clustering or K means produce some Hard clustering. The purpose of the EM clustering is to propose a Soft clustering method.

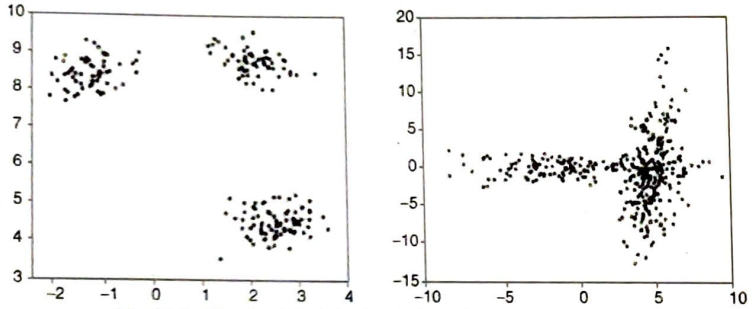


Fig. 6.2.1 : Visual point of view hard clustering and soft clustering

The basic steps for the algorithm are :

1. An initial guess is made for the model's parameters and a probability distribution is made. This is sometimes called the **"E-Step"** for the **"Expected" distribution"**.
2. Newly observed data is fed into the model.
3. The chance distribution of the E-step is tweaked to let in the fresh information. This is sometimes called the "M-step".
4. Steps 2 through 4 are repeated until stability (i.e. a distribution that doesn't change from the E-step to the M-step) is reached.

6.3 Agglomerative Clustering

6.3.1 Affinity

Affinity is a metric function of two arguments with the same dimensionality m.

The most common metrics supported by scikit-learn are :

1. **Euclidean :** Euclid stated that the shortest distance between two points is a line and thus it is predominantly known as **Euclidean distance.** It was often called **Pythagorean metric** since it is derived from the Pythagorean Theorem.

$$d_{Eue} = \sqrt{\sum_{i=1}^{d} |P_i - Q_i|^2}$$

So the distance between two points is

$$\text{dist}((x, y), (a, b)) = \sqrt{(x-a)^2 + (y-b)^2}$$

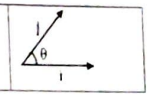
2. **Manhattan or City Block :** The City block distance between two points, C and B, with d dimensions are calculated as

$$d_{CB} = \sum_{i=1}^{d} |P_i - Q_i|$$

The distance between two points measured along axes at right angles. In a plane with p_1 at (x_1, y_1) and p_2 at (x_2, y_2), it is $|x_1 - x_2| + |y_1 - y_2|$.

3. **Cosine distance :** Cosine similarity is a measure of similarity between two vectors. The data objects are treated as vectors. Similarity is measured as the angle θ between the two vectors. Similarity is 1 when θ = 0, and 0 when θ = 90°.

Similarity function is given by

$\cos(i,j) = \frac{i \bullet j}{\|\|i\|\| \times \|\|j\|\|}$	$i \bullet j = \sum_{k-1}^{n} i_k j_k$	$\|\|i\|\| = \sqrt{\sum_{k-1}^{n} i_k^2}$	

Cosine Distance = 1 − Cosine Similarity

$$d_{cosine}(\bar{x}_1, \bar{x}_2) = 1 - \frac{\bar{x}_1 \cdot \bar{x}_2}{||\bar{x}_1||_2 \, ||\bar{x}_2||_2}$$

The cosine distance is useful when we need a distance proportional to the angle between two vectors. If the direction is the same, the distance is null, while it is maximum when the angle is equal to 180° (meaning opposite directions).

6.3.2 Strategy to Aggregate Different Clusters

Different approaches to defining the distance between clusters distinguish the different algorithms, but scikit learn supports the three most common ones i.e. Complete Linkage, Average Linkage and Ward's linkage.

1. **Single-linkage**

- Single Linkage clustering is also called as minimum method, the minimum distance from any object of one cluster to any object of another cluster is considered. In the single linkage method, **D(A,B)** is computed as

  D(A,B) = Min {d(i,j) Where object i is in cluster A and object j is in cluster B}

- This measure of inter-group distance is illustrated in the Fig. 6.3.1.

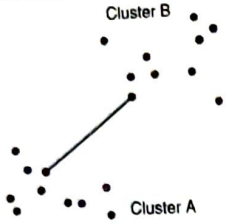


Fig. 6.3.1 : Single-linkage clustering

2. Complete linkage

- **Complete linkage** also called as maximum method, the maximum distance between any object of one cluster to any object of another cluster is considered.
- In the complete linkage method, **D(A,B)** is computed as

  D(A,B) = Max { d(i, j) : Where object i is in cluster A and object j is in cluster B}
- The measure is illustrated in the Fig. 6.3.2.

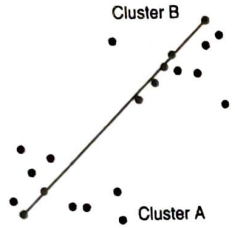


Fig. 6.3.2 : Complete-linkage clustering

3. Average-linkage

- In average-linkage clustering, we consider the distance between any two clusters A and B is taken to be the average of all distances between pairs of objects "i" in A and "j" in B, that is, the mean distance between elements of each cluster.
- In the average linkage method, **D(A,B)** is computed as :

D(A,B) = mean{d(i,j) : Where object i is in cluster A and object j is in cluster B}

- The Fig. 6.3.3 illustrates average linkage clustering.

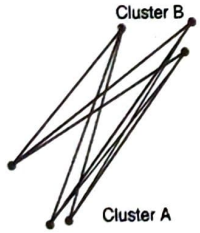
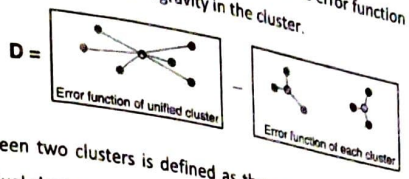


Fig. 6.3.3 : Average-linkage clustering

4. Ward's linkage

- In this method, the error function is defined for every cluster. This error function is the average (RMS) distance of each data-point in a cluster to the centre of gravity in the cluster.



- The distance (D) between two clusters is defined as the error function of the unified cluster minus the error functions of the individual clusters.
- The ward's linkage method computes the distance using formula,

$$\forall C_i, C_j \; L_{ij} = \sum_{x_a \in C_i} \sum_{x_b \in C_j} ||x_a - x_b||^2$$

- The Ward's linkage supports only the Euclidean distance.

5. Centroid clustering

- In centroid method, the distance between two clusters is calculated by finding the distance between two centroids (i.e. mean value of cluster) of the clusters. At every step, two clusters are combined that have minimum centroid distance.
- In the centroid clustering method, D(A,B) is computed as

$$D(A,B) = d(A_{centroid}, B_{centroid})$$

  Where $A_{centroid}$ is the mean value or centroid of cluster A and $B_{centroid}$ is the mean value or centroid of cluster B.
- The Fig. 6.3.4 illustrates centroid clustering.

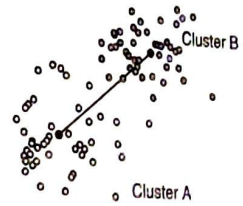


Fig. 6.3.4 : Centroid clustering

6.3.3 Dendrograms

- An agglomerative clustering is typically visualized as a *dendrogram* as shown in Fig. 6.3.5 where each merge is represented by a horizontal line.
- Dendrogram is the aggregation of clusters from bottom to top.
- Initially leaf nodes are single clusters, which are merged till a single root node or cluster generated.

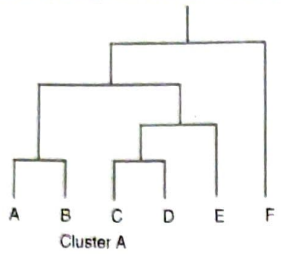


Fig. 6.3.5 : Dendrogram

What is Dendrogram?

- **Dendrogram :** A tree data structure which illustrates hierarchical clustering techniques.
- Each level shows clusters for that level.

  Leaf : Individual clusters

  Root : One cluster
- A cluster at level i is the union of its children clusters at level i + 1.
- A clustering of the data objects is obtained by cutting the dendrogram at the desired level, then each connected component form a cluster.
- Unfortunately, scikit-learn doesn't support the dendrogram but SciPy provides some useful built-in functions.
- The program in Python is given below to create dendrogram.

```
# program to visualize dendrogram
from sklearn.datasets import make_blobs
nb_samples = 25
X, Y = make_blobs(n_samples=nb_samples, n_features=2, centers=3,cluster_std=1.5)
# computing a distance matrix chosen a Euclidean metric
from scipy.spatial.distance import pdist
Xdist = pdist(X, metric='euclidean')
# linkage used is Ward

from scipy.cluster.hierarchy import linkage
Xl = linkage(Xdist, method='ward')
# create and visualize a dendrogram
from scipy.cluster.hierarchy import dendrogram
Xd = dendrogram(Xl)
```

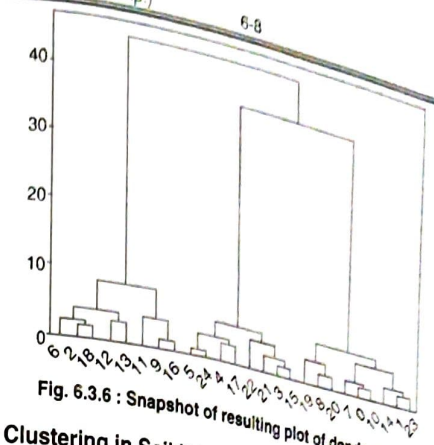



Fig. 6.3.6 : Snapshot of resulting plot of dendrogram

6.3.4 Agglomerative Clustering in Scikit-learn

Consider a more complex dummy dataset with 4 centers.

```
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
#We have now a data set, with 3000 randomized samples around the 4 center points with a standard deviation of
2 for now

nb_samples = 3000
X, _ = make_blobs(n_samples=nb_samples, n_features=2, centers=4,cluster_std=2.0)
# visualize the data set
plt.scatter(X[:,0],X[:,1])
plt.show()
```

A graphical representation is shown in the following Fig. 6.3.7.

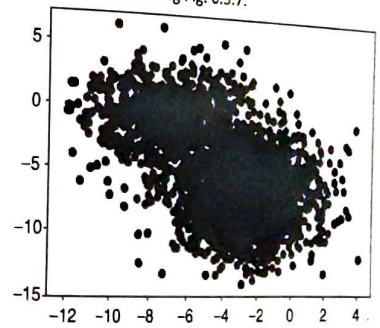


Fig. 6.3.7 : snapshot of resulting plot of data representation

X is the dataset, and y is the label of each data point, according to the sample generation. Agglomerative Clustering uses the method fit_predict() to train the model and transform the original dataset. The number of parameters is set to 4 using the n_clusters parameter while the affinity is set to "Euclidean" (distance between the data points). Finally linkage parameter is set to "complete ".

```
ac = AgglomerativeClustering(n_clusters=4, affinity='euclidean' , linkage='complete')
Y = ac.fit_predict(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```

A graphical representation is shown in the following Fig. 6.3.8.

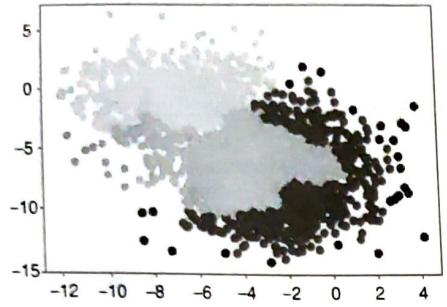


Fig. 6.3.8 : Snapshot of resulting plot of complete linkage

Linkage parameter is set to "average".

```
ac = AgglomerativeClustering(n_clusters=4, affinity='euclidean' ,linkage='average')
Y = ac.fit_predict(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```

The resultant plot is shown in Fig. 6.3.9 :

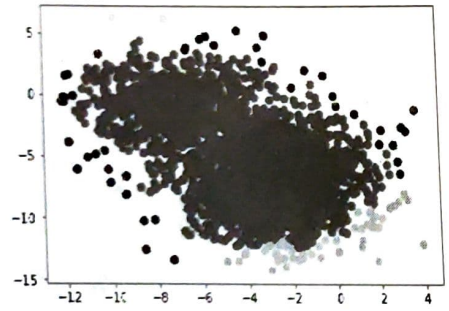


Fig. 6.3.9 : Snapshot of resulting plot of average linkage

The last method, which is often the best (it's the default one), is Ward's linkage, that can be used only with a Euclidean metric (also the default one) :

```
ac = AgglomerativeClustering(n_clusters=4)
Y = ac.fit_predict(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```


The resultant plot is given in Fig. 6.3.10.

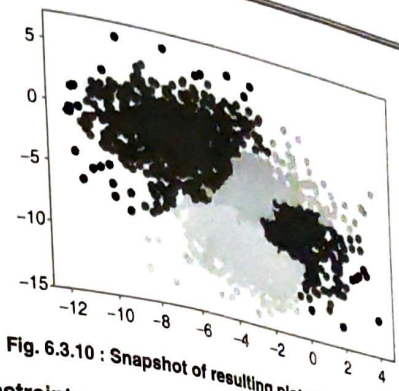


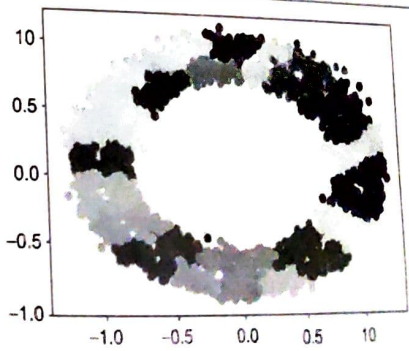
Fig. 6.3.10 : Snapshot of resulting plot of ward linkage

6.3.5 Connectivity Constraints

- An interesting aspect of Agglomerative Clustering is that connectivity constraints which can be added to this algorithm where only adjacent clusters can be merged together and those clusters which are distant i.e. non-adjacent are skipped. Connectivity matrix can be used as a constraint by Scikit-learn and find the clusters to merge. These constraints are useful to impose a certain local structure, but they also make the algorithm faster, especially when the number of the samples is high.
- K-Nearest Neighbour graph function is a common method used.
- Consider circular dummy dataset in the example given below.

```
from sklearn.datasets import make_circles
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
nb_samples = 3000
X, _ = make_circles(n_samples=nb_samples, noise=0.05)

ac = AgglomerativeClustering(n_clusters=20, linkage='average')
ac.fit(X)
plt.scatter(X[:,0],X[:,1], c=ac.labels_, cmap='rainbow')
```



```
# Now we can try to impose a constraint with different values for k:
from sklearn.neighbors import kneighbors_graph
acc = []
k = [50, 100, 200, 500]
for i in range(4):
    kng = kneighbors_graph(X, k[i])
    ac1 = AgglomerativeClustering(n_clusters=20, connectivity=kng,linkage='average')
    ac1.fit(X)
    plt.scatter(X[:,0],X[:,1], c=ac1.labels_, cmap='rainbow')
    acc.append(ac1)
```

Imposing a constraint based on K-Nearest Neighbours, allows controlling how the agglomeration creates new clusters and can be a powerful tool for tuning the models, or for avoiding elements whose distance is large in the original space could be taken into account during the merging phase. It is helpful in clustering the images.

6.4 Introduction to Recommendation Systems

- Recommendation system is used to predict future preferences of a set of items and also recommend the top n items to users.
- Many e-commerce websites uses recommendation system to increase their sales by using the power of data.
- This system finds the user's interest to recommend items.
- Traditionally, there are two methods to construct a recommendation system :
  1. User or Content-based recommendation
  2. Collaborative Filtering

1. **Content-based recommendation system:** It makes recommendations using a user's item and profile features. To find the user's future interest, the system needs the information about user's past interest in the items. So only similar items can be grouped together based on their features.
2. **Collaborative Filtering systems:** It filters the items in which we are interested. This system is based on the assumption that if user1 likes item X and user2 likes the same item X plus another item Y, then user1 may be interested in item Y. So system can use historical data to predict new interactions.

6.4.1 Naïve User based Systems

- Let us consider a set of users represented by feature vectors :

$$U = \{\bar{u}_1, \bar{u}_2, ..., \bar{u}_n\} \text{ where } \bar{u}_n \in \mathbb{R}^n$$

- Assume the features are age, gender, interests, and so on. Set of item I is :

$$I = \{i_1, i_2, ..., i_m\}$$

- Every user is associated with a subset of items or items can be viewed or feedback has been taken for those items. So there is a relationship between user and items.

$$g(\bar{u}) \rightarrow \{i_1, i_2, ..., i_k\} \quad \text{where } k \in (0, m)$$

- In a user-based system, the users are periodically clustered (normally using a **K-Nearest Neighbour** approach), and therefore, considering a generic user u (also new), we can immediately determine all the users who are similar (therefore neighbours) to our sample:

$$B_R(\bar{u}) = \{\bar{u}_1, \bar{u}_2, ..., \bar{u}_k\}$$

- At this point, we can create the set of suggested items using the relation previously introduced :

$$I_{suggested}(\bar{u}) = \left\{ \bigcup_i g(\bar{u}_i) \text{ where } \bar{u}_i \in B_R \bar{u} \right\}$$

- This set contains all the unique products positively rated or bought by the neighbourhood.

6.4.2 Content Based Systems

- Recommendations are based on information on the content of items rather than on other users' opinions.
- Uses machine learning algorithms to induce a profile of the user's preferences from examples based on content features.
  - No need for data about other users.
  - No cold-start or sparsity problems.
  - Able to recommend to users with unique tastes.
  - No first-rater problem.
- **Cold Start :** enough users in the system to find a match.
- **Sparsity :** The user/ratings matrix is sparse, and it is hard to find users that have rated the same items.
- **First Rater :** Not for an item that has not been previously rated.
- This is one of the simplest methods which is based on products and modelled as feature vectors :

$$I = \{\bar{I}_1, \bar{I}_2, ..., \bar{I}_n\} \text{ where } \bar{I}_n \in \mathbb{R}^n$$

- The features can also be categorized like users. For example, the type of a book or a movie, and they can be used together with numerical values (like price, length, number of positive reviews, and so on) after encoding them.

6.4.3 Model Free Collaborative Filtering

- It is based on rating among the users and "lightweight" filtering approach. It is computed in-memory without the use of complex algorithms like ALS (Alternating Least Squares) which executes in parallel environment.
- Assume that we have M products and N users, then user-preference sparse matrix is given below.

$$U_{pref} = \begin{pmatrix} u_0^{(0)} & \cdots & u_0^{(M)} \\ \vdots & \ddots & \vdots \\ u_N^{(0)} & \cdots & u_N^{(M)} \end{pmatrix}$$

- Each element from the above represents the rating given by the user i to the item j.
- 0 means there is no rating

- Two kinds of ratings or feedbacks
  1. **Explicit feedback :** An actual rating (like 3 stars or 8 out of 10)
  2. **Implicit feedback :** A page view, song play, move play and so forth
- After the user preference matrix, compute the affinity matrix based on a similarity metric using Euclidean as given follows :

$$A = \begin{pmatrix} d(u_0,u_0) & \cdots & d(u_N,u_0) \\ \vdots & \ddots & \vdots \\ d(u_0,u_N) & \cdots & d(u_N,u_N) \end{pmatrix}$$

- For explicit feedback, the Euclidean metric is preferable, but when it's implicit (like page views, song/movie plays, etc.), it has a different meaning and need another metric function. For example, if we have a binary encoding like 0 and 1 (0 means no interaction and 1 means single or multiple interaction), then Hamming distance is more appropriate. Our goal is to cluster the users according to their rating. After clustering, we can check which products have higher rating for a given user and therefore are more likely to be bought.
- Once the top neighbors determined, union of most rated neighbour products is used to suggest the list of top neighbours :

$$P_{suggest}(i) = \left\{ \bigcup_{j \neq i} P(j) \right\}$$

The code to create a distance matrix using sklearn for 10 users and 10 items.

```
from scipy.sparse import dok_matrix
from sklearn.metrics.pairwise import pairwise_distances
import numpy as np

# Set random seed (for reproducibility)
np.random.seed(50)

# Create a dummy user-item dataset
nb_users = 10
nb_products = 10
max_rating = 5
max_rated_products = 8

X_preferences = dok_matrix((nb_users, nb_products), dtype=np.uint8)

for i in range(nb_users):

  # Extract n random products
```



```
  n_products = np.random.randint(0, max_rated_products+1)
  products = np.random.randint(0, nb_products, size=n_products)

  # Populate preference sparse matrix
  for p in products:
    X_preferences[i, p] = np.random.randint(0, max_rating+1)

# Compute pairwise distances
distance_matrix = pairwise_distances(X_preferences, metric='euclidean')

# Sort distances
sorted_distances = np.argsort(distance_matrix, axis=1)
print(sorted_distances)
```

```
[[0 1 5 7 4 8 6 2 9 3]
 [0 1 5 7 4 8 6 2 9 3]
 [2 0 1 5 7 4 8 3 6 9]
 [3 2 4 6 0 1 5 7 9 8]
 [4 0 1 5 7 8 6 2 9 3]
 [0 1 5 7 4 8 6 2 9 3]
 [6 0 1 5 7 4 8 9 2 3]
 [0 1 5 7 4 8 6 2 9 3]
 [8 0 1 5 7 4 9 6 2 3]
 [9 8 0 1 5 6 7 4 2 3]]
```

```
test_user=5

# Take the top-5 simular users
for d in sorted_distances[test_user][::-1][0:5]:
  print(d)
```

Output

```
3
9
2
6
8
```

6.4.4 Singular Value Decomposition

- Singular value decomposition (SVD) is a method of decomposing a matrix into three other matrices.
- Given any $m \times n$ matrix **A**, algorithm to find matrices **U**, **V**, and **W** such that

$$A = U W V^T$$

U is $m \times n$ and orthonormal

W is $n \times n$ and diagonal

V is $n \times n$ and orthonormal

$$( ) = ( ) \begin{pmatrix} {}_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & {}_n \end{pmatrix} ( )$$

- +By assuming m ≥ n rewrite above equation using summation notation

$$a_{ij} = \sum_{k=1}^{n} u_{ik} s_k v_{jk}$$

- Simplify the expression into a single summation. The variables, $\{s_i\}$, are called singular values and are normally arranged from largest to smallest.

$$s_{i+1} \leq s_i$$

- The columns of U are called left singular vectors, while those of V are called **right singular vectors.**
- We know that U and V are orthogonal, that is:

$$U^T U = V V^T = I$$

Where I is the identity matrix (the diagonals of the identity matrix are 1, other values being 0).

- svd() function calculate SVD.
- svd() function takes matrix as input and returns U, Sigma and V^T
- The following example defines a 3 × 2 matrix and calculates the Singular-value decomposition.

```
# Singular-value decomposition
from numpy import array
from scipy.linalg import svd
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# SVD
U, s, VT = svd(A)
print(U)
print(s)
print(VT)
```

Output

```
A matrix [[1 2]
         [3 4]
         [5 6]]
```



```
U matrix
[[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]

S matrix
[9.52551809 0.51430058]

V^T matrix
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
```

6.4.5 Alternating Least Squares

- Alternating Least Squares (ALS) is a model use to fit our data and find similarities.
- Least square optimization problem is defined as a loss function to find the latent factor as given below.

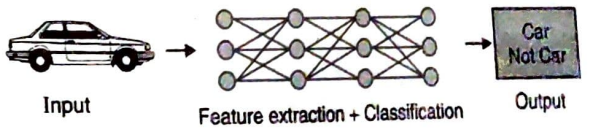
$$L = \sum_{(i,j)} \left(r_{ij} - \bar{p}_i \cdot \bar{q}_j^T \right)^2 + \alpha \left(||\bar{p}_i||^2 + ||\bar{q}_j||^2 \right)$$

- L can be identified by using known samples (user, item).
- The second term of the above equation is a regularization factor.
- Any optimization method can be used to solve the whole problem.
- Two main iterating steps of algorithm are :
  1. p_i is fixed and q_j is optimized
  2. q_j is fixed and p_i is optimized

6.5 Fundamentals of Deep Networks

- A machine learning subfield of learning representations of data. Exceptionally effective at learning patterns.
- Deep learning algorithms attempt to learn (multiple levels of) representation by using a hierarchy of multiple layers.
- If you provide the system, tons of information, it begins to understand it and respond in useful ways.

(a) Machine learning



(b) Deep learning

Fig. 6.5.1

6.5.1 Defining Deep Learning

'Deep Learning' means using a neural network with several layers of nodes between input and output. It is the series of layers between input and output for feature identification and processing in a series of stages, just as our brains seem to.

Uses of Deep Learning

1. Annually designed features are incomplete and takes long time, but learned features are adaptable and fast to learn.
2. Deep learning provides a very flexible, (almost?) Universal, a learnable framework for representing the world, visual and linguistic information. Can learn both unsupervised and supervised.
3. Effective end-to-end joint system learning utilize large amounts of training data.

6.5.2 Common Architectural Principles of Deep Networks

How do We Train Deep Architectures?

- It's an inspiration from mammal brain.
- Deep network has Multiple Layers of "neurons" (Rumelhart et al 1986).
- Training for each layer is required to learn higher level abstraction.
- **Example** : Pixels → Edges → Contours → Object parts → Object categories → Local Features → Global Features
- Greedy strategy is also used to train the layers one-by-one (Hinton et al 2006)

6.5.2(A) Multilayer Perceptron with Back-propagation : First Deep Learning Model

(Rumelhart, Hinton, Williams 1986)

Multilayer Perceptrons with Back-propagation
First deep learning model (Rumelhart, Hinton, Williams 1986)

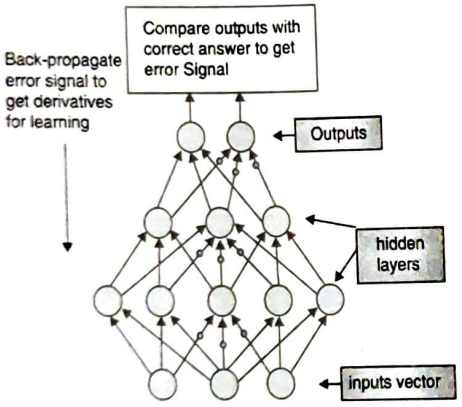


Fig. 6.5.2

Drawbacks of Back-propagation based Deep Neural Networks

- They are discriminative models.
- All the information is collected from labels.
- But the labels are not able to give all information.
- It requires a large amount of labeled data.
- Gradient descent with random initialization leads to poor local minima.

6.5.2(B) Deep Belief Networks (Hinton et al. 2006)

- Pre-training of network is required before using back-propagation.
- Use the weights of the pre-trained network as the initial point for the traditional back-propagation.
- This leads to quicker convergence to a better solution.
- Pre-training is fast; fine-tuning can be slow.

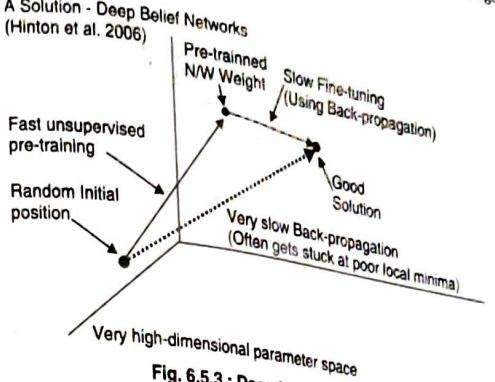


Fig. 6.5.3 : Deep belief networks

6.5.2(C) Five Popular and Widely-used Deep Learning Architectures

1. Convolutional Neural Networks
2. Recurrent Neural Networks
3. Autoencoders
4. Generative Adversarial Networks
5. ResNets

6.5.2(D) Building Blocks of Deep Networks

- **Activation functions :** linear, ReLU, sigmoid, tanh, etc.
- **Layers :** Fully connected, convolutional and pooling, recurrent (Problem: The graph is not acyclic. To compute the gradients, we unroll the computational cycle over timesteps, and backpropagate through this structure.), resnet (skip connections over layers), etc.

Review Questions

Q.1 Explain Hierarchical Clustering basic algorithm.

Q.2 Explain Expectation Maximization Clustering algorithms in detail.

Q.3 What are various strategy to Aggregate Different Clusters.

Q.4 Explain Model Free Collaborative Filtering.

Q.5 Write short note on Recommendation Systems

Q.6 What is Deep Learning? Explain.

□□□